(19)日本国特許庁（ＪＰ）　(12) **公開特許公報（Ａ）**　(11)特許出願公開番号

特開平9－22362

(43)公開日　平成９年(1997)１月21日

(51)Int.Cl.[6]	識別記号	庁内整理番号	ＦＩ		技術表示箇所
Ｇ０６Ｆ　9/45		9189－5B	Ｇ０６Ｆ　9/44	３２２Ｈ	
		9189－5B		３２２Ｆ	

審査請求　未請求　請求項の数６　ＦＤ　（全 16 頁）

(21)出願番号　特願平8－99596

(22)出願日　平成８年(1996)３月29日

(31)優先権主張番号　０８／４１４２６７

(32)優先日　1995年３月31日

(33)優先権主張国　米国（ＵＳ）

(71)出願人　591064003
サン・マイクロシステムズ・インコーポレーテッド
SUN MICROSYSTEMS, INCORPORATED
アメリカ合衆国　94043　カリフォルニア州・マウンテンビュー・ガルシア　アヴェニュウ・2550

(72)発明者　ダニエル・ディ・グローヴ
アメリカ合衆国　94043　カリフォルニア州・マウンテンビュー・ルース　アヴェニュ・345

(74)代理人　弁理士　山川　政樹

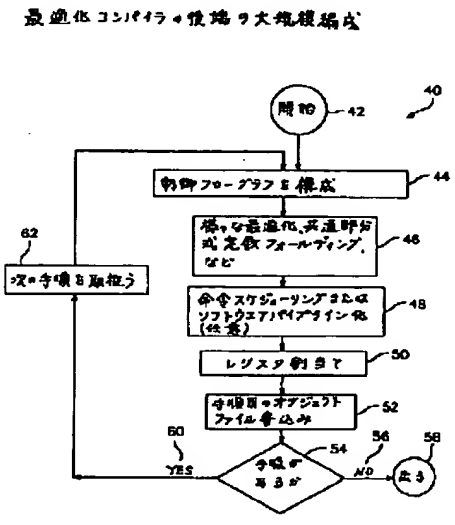
最終頁に続く

(54)【発明の名称】　コンピュータシステムおよびコンピュータ制御方法

(57)【要約】

【課題】　ターゲットコンピュータプログラムの一部内の生変数集合をより正確にかつ効率的に確認して、コンピュータの中央演算処理装置内のメモリレジスタをより正確に割り当てるための最適化コンパイラのプロセスおよび方法

【解決手段】　本発明は、制御フローグラフにファイ関数を付加するステップを含む、コンピュータプログラムへの静的単一割当て変換を実行するステップを含む。さらに、変数の使用を表す基本ブロックを、ファイ関数とファイ関数に収束する変数の定義との間で制御フローグラフに付加する。次いで、逆方向データフロー解析を実行して、生変数集合を確認する。ファイ関数の引数内の変数は、このデータフロー解析ではそれらの変数の使用として含まれない。データフロー解析は、生変数集合が反復の間で一定になるまで反復して実行される。



1

【特許請求の範囲】

【請求項１】　中央演算処理装置（ＣＰＵ）と、前記ＣＰＵに結合されたランダムアクセスメモリ（ＲＡＭ）とを有し、固定した数のＣＰＵレジスタを有するターゲットコンピュータアーキテクチャ上で動作するように目的プログラムをコンパイルする際に使用するコンピュータシステムにおいて、

前端コンパイラとコードオプティマイザと後端コード生成器とを有して前記コンピュータシステム内に常駐するコンパイラシステムを含み、

前記コードオプティマイザは目的プログラム中の基本ブロック用の生変数の集合を決定するように構成され、前記ターゲットコンピュータアーキテクチャ内の前記固定した数のＣＰＵレジスタを、その固定した数のＣＰＵレジスタのうちの一つのレジスタ内ではなくコンピュータメモリ内に記憶する必要のある前記生変数の集合内の変数の数が最小となるように、前記生変数の集合に割り当てることができ、前記コードオプティマイザが、制御フローグラフの少なくとも一部に、前記制御フローグラフ中の変数の使用を表すダミーブロックをファイ関数ノードの前に挿入し、かつ前記ダミーブロックを考慮することによって生変数の集合を決定するコンピュータシステム。

【請求項２】　複数のＣＰＵレジスタを有するターゲットコンピュータアーキテクチャ上で動作するように目的プログラムをコンパイルするコンパイラシステムにおいて、

前記目的プログラムのソースコードを入力として受け取り、かつ対応する中間コード集合を出力するように構成された前端部分と、

前記前端部分に結合され、かつ前記中間コードの集合を入力として受け取りかつ第２の中間コードの集合を出力するように構成されたコードオプティマイザとを含み、

前記第２の中間コードの集合が、前記ターゲットコンピュータアーキテクチャの前記複数のＣＰＵレジスタを、前記複数のＣＰＵレジスタ内ではなくコンピュータメモリ内に記憶する必要のある前記生変数の集合内の変数の数が最小となるように、前記目的プログラム中の基本ブロックの生変数の集合に割り当てる前記目的プログラム用のコードを含み、前記コードオプティマイザが、制御フローグラフの少なくとも一部に、前記制御フローグラフ中の変数の使用を表すダミーブロックをファイ関数ノードの前に挿入し、かつ前記ダミーブロックを考慮することによって生変数の集合を決定するコンパイラシステム。

【請求項３】　複数のＣＰＵレジスタを有するターゲットコンピュータアーキテクチャ上で動作するように目的プログラムをコンパイルするコンパイラシステムで使用するためのコードオプティマイザにおいて、

前記目的プログラムの中間コード表現を入力として受け取るように構成された第１の部分と、

前記第１の部分に結合され、かつ前記目的プログラムの基本ブロックの生変数の集合を決定するように構成された第２の部分と、

前記第２の部分に結合され、かつ前記ターゲットコンピュータアーキテクチャの前記複数のＣＰＵレジスタを、前記複数のＣＰＵレジスタ内ではなくコンピュータメモリ内に記憶する必要のある前記生変数の集合内の変数の数が最小となるように、前記目的プログラムの基本ブロックの生変数の集合に割り当てるように構成された第３の部分とを含み、その第３の部分が、制御フローグラフの少なくとも一部に、前記制御フローグラフ中の変数の使用を表すダミーブロックをファイ関数ノードの前に挿入し、かつ前記ダミーブロックを考慮することによって生変数の集合を決定するコードオプティマイザ。

【請求項４】　目的プログラム中の変数の生集合を、固定した数のＣＰＵレジスタのうちの一つのレジスタ内ではなくコンピュータメモリ内に記憶する必要のある前記生変数の集合内の変数の数が最小となるように、ターゲットコンピュータアーキテクチャ内の前記固定した数のＣＰＵレジスタに割り当てるコンピュータ制御方法において、

前記目的プログラムの手順を表す干渉グラフを構成するステップと、

前記目的プログラムの制御フローグラフにファイ関数を付加するステップを含む、前記目的プログラムの前記手順の静的単一割当て変換を実行するサブステップと、

制御フローグラフ中の変数の使用を表す使用ブロックを、ファイ関数と前記変数の定義を含むブロックとの間に挿入するサブステップと、

制御フローグラフの逆方向データフロー解析によって目的プログラム中の基本ブロックの変数の前記生集合を決定するサブステップと、

変数の生集合を固定した数のＣＰＵレジスタにマップするサブステップとによって目的プログラムの変数の生集合を決定するステップとを含む方法。

【請求項５】　固定した数のＣＰＵレジスタを有するターゲットコンピュータアーキテクチャ上で動作するようにコンパイルされた目的プログラムの２進コードを最適化するコンピュータ制御方法において、

前記目的プログラムのソースコードを受け取り、かつ前記ターゲットコンピュータアーキテクチャ上で処理できる前記目的プログラムを表す２進コードを出力するように構成され、前端部分とコードオプティマイザ分と後端コード生成器を含むコンパイラシステムを用意するステップと、

前記コンパイラシステムの前記前端部分から中間コードを受け取り、かつ前記目的プログラムを表す前記中間コード内の変数の生集合を前記ターゲットコンピュータアーキテクチャ内の前記固定した数のＣＰＵレジスタに、

前記固定した数のＣＰＵレジスタ内ではなくメモリ内に記憶しなければならない変数の前記生集合内の変数の数が最小となるように、割り当てるように構成された前記コンパイラシステムのコードオプティマイザを用意するステップとを含み、前記目的プログラムを表す前記中間コード内の変数の生集合を前記固定した数のＣＰＵレジスタに割り当てる際に、

前記目的プログラムの手順を表す干渉グラフを構成し、

前記目的プログラムの制御フローグラフにファイ関数を付加するステップを含む、前記目的プログラムの前記手順の静的単一割当て変換を実行するサブステップと、

制御フローグラフ中の変数の使用を表す使用ブロックを、ファイ関数と前記変数の定義を含むブロックとの間に挿入するサブステップと制御フローグラフの逆方向データフロー解析によって目的プログラム中の基本ブロックの変数の前記生集合を決定するサブステップとによって目的プログラムの変数の生集合を決定し、

変数の生集合を固定した数のＣＰＵレジスタにマップし、

前記固定した数のＣＰＵレジスタ内ではなくメモリ内に記憶される必要のある変数の前記生集合内の変数の数が最小となるような、前記目的プログラム中の変数の前記生集合の、前記ターゲットコンピュータアーキテクチャ内の前記固定した数のＣＰＵレジスタへの割当てを含む、前記目的プログラムの第２の中間コードバージョンを前記後端コード生成器に出力することによって実行され、それによって前記目的プログラムの２進コードが最適化される方法。

【請求項６】　制御フローグラフの少なくとも一部に、前記制御フローグラフ中の変数の使用を表すダミーブロックをファイ関数ノードの前に挿入するステップと、

前記ダミーブロックを考慮することによって生変数の集合を決定するステップとを含む、目的プログラムの生変数の集合を決定するコンパイラのコードオプティマイザ分を改善するコンピュータ制御方法。

【発明の詳細な説明】

【０００１】

【発明の属する技術分野】本発明は、コンピュータシステムの最適化コンパイラの分野に関する。さらに具体的には、最適化コンパイラのコード最適化パスの間に使用可能な中央演算処理装置（「ＣＰＵ」）レジスタに生変数を割り振るための改善された方法および装置に関する。

【０００２】

【従来の技術】コンピュータプログラムは、それらの実行時間中およびメモリ使用中にできるだけ効率的であることが望ましい。このような要求から最適化コンパイラが開発されることになった。最適化コンパイラは、一般に、コンパイラ前端とコンパイラ後端の間にあるコードオプティマイザを含む。コードオプティマイザは、入力として「中間コード」出力をコンパイラ前端によって受け取り、このコードに基づいてそれに対して様々な変換を実行するように動作し、その結果より迅速かつ効率的な目的プログラムが得られる。変換されたコードは、コンパイラ後端に送られ、次いでコンパイラ後端がそのコードを関係する特定の機械（すなわちＳＰＡＲＣ、Ｘ８６、ＩＢＭなど）用に２進形に変換する。コードオプティマイザ自体は、できるだけ迅速かつメモリ効率が高い必要がある。

【０００３】例えば、たいていのコードオプティマイザは、ターゲットコンピュータ内の使用可能なＣＰＵレジスタの使用を最適化しようと試みる。これは、グラフ作成技法やフロー制御技法を使用して、目的プログラムの「生変数集合」を決定し、かつプログラム実行中にレジスタからメモリ内に「スピル（ｓｐｉｌｌ）する」必要のある変数の数が最小となるように、プログラムコードを再配置することによって行われる。このレジスタ割当てプロセスは、コンパイルされる普通の目的プログラムでは数百個の変数に基づいて動作する必要があり、また科学の目的プログラムでは１万～２万個もの変数が必要となる。したがって、このような多数の目的プログラム変数を処理する場合、コードオプティマイザ自体は、過度に時間を浪費し、また最適化計算を正確に処理しなければ、これらのレジスタ割当て計算を行う際にメモリを過度に使用する恐れがある。

【０００４】過去において、一般的には最適化コンパイラを開発する試み、具体的にはそれ自体できるだけ効率的に動作するコードオプティマイザ・モジュールを開発する試みが行われてきた。最適化コンパイラと使用する関連技法についての一般的な検討は、Ａｌｆｒｅｄ Ｖ．Ａｈｏ，Ｒａｖｉ Ｓｅｔｈｉおよび Ｊｅｆｆｒｅｙ Ｄ．Ｕｌｍａｎ著の教科書「Ｃｏｍｐｉｌｅｒｓ：Ｐｒｉｎｃｉｐｌｅｓ，Ｔｅｃｈｎｉｑｕｅｓ ａｎｄ Ｔｏｏｌｓ」Ａｄｄｉｓｏｎ－ＷｅｓｌｅｙＰｕｂｌｉｓｈｉｎｇ Ｃｏ １９８８，ＩＳＢＮ ０－２０１－１００８８－６、特に９章および１０章、５１３～７２３頁に記載されている。コードオプティマイザの計算時間を短縮するそのような試みの一つは、レジスタ割当てプロセスにおいて「生変数集合」を計算する時間を短縮することである。通常、制御フローグラフ中の生変数の計算には、N^2 回の計算とメモリ記憶空間が必要となる。ただし、Ｎは関係する変数の数である。計算時間を短縮するこのような試みでは、関係する「生変数集合」を見つけるための方法は、目的の静的単一割当て（「ＳＳＡ」）グラフを構成し、かつ「Φ関数」または制御フローパスが結合し、かつ異なる値が融合する「Φノード」と呼ばれる特殊な定義をグラフに挿入することを必要とした。この方法では、変数の数の計算がN^2 回からＮ回にまで減少し、時間とメモリ使用量が大幅に減少した。この方法は、参照することにより本明細書の一部と

なる、Ｐｒｅｓｔｏｎ　ＢｒｉｇｇｓのＰｈＤ．論文「Ｒｅｇｉｓｔｅｒ　Ａｌｌｏｃａｔｉｏｎ　ｖｉａ　Ｇｒａｐｈ　Ｃｏｌｏｒｉｎｇ」Ｒｉｃｅ　Ｕｎｉｖｅｒｓｉｔｙ、１９９２年４月に記載されている。しかしながら、Ｂｒｉｇｇｓのこの方法では、「生集合」中のいくつかの変数が、実際には可能であっても、レジスタ内で他の変数と共存できない（したがって変数がメモリ内にスピルされる）と考えられるので、変数の「スピル」が必要以上に多くなる。

【０００５】

【発明が解決しようとする課題】本発明では、簡潔な方法を使用して、これらの誤った変数のスピルを減らし、それによってコードオプティマイザ自体の効率を低下させることなく目的プログラムの効率を上げることを目的としている。

【０００６】本発明の目的、特徴および利点は、以下の説明からさらに明らかとなろう。

【０００７】

【課題を解決するための手段】本発明は、最適化コンパイラ内で機械レジスタを目的プログラムの「生変数集合」に割り振る際に変数のスピルを最小限に抑えるための経済的、高性能、適応性のあるシステムおよび方法を提供することによって、上述のシステムの欠点を克服する。好ましい実施態様では、変数定義ノードの直後に、ダミーブロック（「使用」ブロックと呼ぶ）をフロー制御グラフ中に挿入する。次いで、これらの「使用」ブロックを検討し「Φ関数」は検討しないように、プロセスを決定する「生変数集合」を変更する。この変更されたプロセスは、上述の誤った変数のスピルを最小限に抑える効果がある。

【０００８】本発明の一態様では、目的プログラムの基本ブロックの生変数の集合を決定し、かつ変数のスピルが最小になるようにターゲットコンピュータアーキテクチャのＣＰＵレジスタを生変数の集合に割り振るように構成された部分を有し、制御フローグラフの少なくとも一部に、前記制御フローグラフ中の変数の使用を表すダミーブロックをファイ関数ノードの前に挿入し、かつ前記ダミーブロックを検討することによって生変数の集合を決定する、コンパイラシステムで使用するためのコードオプティマイザが提供される。

【０００９】本発明の他の態様では、目的プログラムを表す第２の中間コードを生成するコードオプティマイザが、前記プロセスに従って目的プログラムの生変数集合を発生して、この生変数集合をターゲットコンピュータアーキテクチャの指定されたＣＰＵレジスタにマップし、ファイ関数ノードの前に制御フローグラフ中の変数の使用を表すダミーブロックを前記制御フローグラフの少なくとも一部に挿入し、かつ前記ダミーブロックを検討することによって生変数の集合を決定するコンパイラシステムが提供される。

【００１０】本発明の他の態様では、前記プロセスに従って目的プログラムの生変数の集合を決定し、かつこの生変数集合を指定されたＣＰＵレジスタにマップするように構成され、制御フローグラフの少なくとも一部に、前記制御フローグラフ中の変数の使用を表すダミーブロックをファイ関数ノードの前に挿入し、かつ前記ダミーブロックを検討することによって生変数の集合を決定するコードオプティマイザを含むコンパイラシステムを含む、固定した数のＣＰＵレジスタを有するターゲットコンピュータアーキテクチャ上で動作するように目的プログラムをコンパイルする際に使用するコンピュータシステムが提供される。

【００１１】本発明の他の態様では、制御フローグラフの少なくとも一部に、前記制御フローグラフ中の変数の使用を表すダミーブロックをファイ関数ノードの前に挿入し、かつ前記ダミーブロックを検討することによって生変数の集合を決定する、コンパイラ中の生変数の集合をＣＰＵレジスタの指定された集合に割り振って、目的プログラムの動作を最適化するコンピュータ制御方法が提供される。

【００１２】

【発明の実施の形態】表記法および用語以下の詳細な説明は、主として、コンピュータメモリ内のデータビットに基づく動作の手順および記号表現の形で表されている。これらの手順の説明および表現は、データ処理技術分野の熟練者が彼らの仕事の要旨を当分野の他の熟練者に最も効率的に伝えるのに使用する手段である。

【００１３】本願では、手順は、一般に、所望の結果が得られる自己矛盾のないステップの連続だと考える。これらのステップは物理的量の物理操作を要するものである。通常、必ずしもそうではないが、これらの量は、記憶、転送、組合せ、比較、それ以外の操作が可能な電気信号や磁気信号の形をとる。主として通常の使用の理由から、これらの信号をビット、値、要素、記号、文字、用語、番号などと呼ぶと便利なことが多い。しかしながら、これらおよび同様な用語はすべて、適切な物理量に関連し、かつこれらの量に適用された便利な表示に過ぎないことに留意されたい。

【００１４】さらに、行われる操作は、加算や比較など、通常人間のオペレータが行う頭脳操作に関連する用語で呼ばれることが多い。人間のオペレータのそのような能力は、本発明の一部を成す本願に記載されているどの操作においても不要であり、たいていの場合望ましくない。すなわち、操作は機械操作である。本発明の操作を実行する有用な機械は、汎用デジタルコンピュータまたは同様な装置である。すべての場合に、コンピュータを操作する際の方法操作と計算の方法自体の相違に留意されたい。本発明は、電気物理信号や他の物理信号（例えば、磁気物理信号、化学物理信号）を処理して他の所望の物理信号を発生させる際にコンピュータを操作する

方法ステップに関する。

【００１５】本発明は、これらの操作を実行する装置にも関する。この装置は、所要の目的用に特別に構成されている。すなわち、コンピュータ内に記憶されたコンピュータプログラムによって選択的に活動化されたり再構成される汎用コンピュータを含んでいる。本願に提示した手順は、本来、特定のコンピュータまたは他の装置に関するものではない。特に、本願の教示に従って書き込まれたプログラムとともに、様々な汎用機械が使用できる。すなわち、所要の方法ステップを実行するように、より特殊化された装置を構成するほうが便利である。これらの様々な機械に必要とされる構造は、以下の記述から明らかとなろう。

【００１６】最適化コンパイラのコード最適化パスの際に使用可能な中央演算処理装置（「ＣＰＵ」）レジスタに生変数を割り振り、変数のレジスタ「スピル」を最小限に抑える装置および方法を開示する。以下の記述では、説明を目的として、本発明を完全に理解できるように、特定の命令呼出し、モジュールなどを示してある。しかしながら、本発明がこれらの特定の詳細なしに実行できることは当業者には明らかであろう。他の例では、本発明が不必要にあいまいにならないように、周知の回路および装置をブロック図形式で示してある。同様に、好ましい実施形態では、ユニプロセッサコンピュータシステムおよびマルチプロセッサコンピュータシステムならびにソラリスオペレーティングシステムを利用する。これらはすべてＳｕｎ　Ｍｉｃｒｏｓｙｓｔｅｍｓ，Ｉｎｃ．が製造および販売している。しかしながら、本発明は、他のコンピュータハードウェアシステム上でも実施でき、また他の互換なオペレーティングシステムを使用しても実施できる。

【００１７】動作環境

本発明が使用される環境は、広く一般に普及したコンピュータシステムを取り巻いている。そこでは、汎用コンピュータやワークステーションやパーソナルコンピュータが、クライアントサーバ配置内で様々な形式の通信リンクを介して接続され、システムの他の部材によって実行およびアクセスできるように、プログラムおよびデータが、多くはオブジェクトの形で、システムの様々な部材によって使用可能となっている。汎用ワークステーションコンピュータのいくつかの要素を図１に示す。図１には、入出力（「Ｉ／Ｏ」）部２、中央演算処理装置（「ＣＰＵ」）３、およびメモリ部４を有するプロセッサ１が示されている。入出力（「Ｉ／Ｏ」）部２は、キーボード５、ディスプレイ装置６、ディスク記憶装置９、およびＣＤ－ＲＯＭ駆動装置７に接続される。ＣＤ－ＲＯＭ駆動装置７は、一般にプログラム１０とデータを含むＣＤ－ＲＯＭ媒体８を読むことができる。図２は、前端コンパイラ２４、コードオプティマイザ２６、および後端コード生成器２８を含む、代表的な最適化コンパイラ２０を示す。コンパイラの前端２４は、ソース言語２２内に書き込まれたプログラムを入力として受け取り、この言語に対して様々な字句、構文および意味の解析を実行し、目的プログラムを表すコード３２の中間集合を出力する。この中間コード３２は、迅速に動作する機械コードが得られるように中間コードを改善しようと試みるコードオプティマイザ２６モジュールへの入力として使用される。コードオプティマイザ２６の中のいくつかは些細なものであり、他のものは様々な計算を行って、可能な最も効率的な目的プログラムをつくり出そうと試みる。この後者のタイプは、「最適化コンパイラ」と呼ばれ、共通の部分式の削除、無意味なコードの削除、一時変数の名前変更、および２つの独立した隣接するステートメントの交換ならびにレジスタ割当てなどのコード変換を含む。

【００１８】図３は、最適化コンパイラ４０の代表的な編成を示す。中間コードのエントリ上に（４２）制御フローグラフを構成する（４４）。この段階で、上述のコード変換（共通の部分式の削除、無意味なコードの削除、一時変数の名前変更、および２つの独立した隣接するステートメントの交換など）を行う（４６）。命令のスケジューリングまたは「パイプライニング」はこの時点で行う（４８）。次いで「レジスタ割当て」を実行し（５０）、変更されたコードを書き出して、コンパイラ後端がそのコードをターゲット機械（すなわちＳＰＡＲＣ，Ｘ８６など）の２進言語に変換する（５８）。出願者の発明の焦点は、この「レジスタ割当て」５０のプロセスである。

【００１９】レジスタ割当てたいていの従来のコンピュータの中央演算処理装置（ＣＰＵ）は、一般に、迅速なアクセスのために様々なデータが記憶される一組のレジスタを含む。コンピュータプログラムの実行中、プログラム中に存在する変数の値は、普通、これらのレジスタ内に記憶される。現代のＣＰＵは、普通、３２～６４個の浮動小数点レジスタを所有する。多くの業務関連アプリケーションプログラムには十分であるが、科学用途に関連するプログラムは、通常、使用可能なレジスタの数よりもはるかに多くの変数を使用する。プログラム中に存在する変数が多すぎる場合、過剰な変数の値は、普通、キャッシュメモリ内に記憶されるか、あるいは従来のランダムアクセスメモリ内に記憶される。キャッシュ内かまたは従来のメモリ内に記憶される変数の値へのアクセスには、ＣＰＵのレジスタに記憶される値にアクセスするよりもはるかに多くの時間がかかる。一般に、キャッシュメモリ内に記憶される変数へのＣＰＵアクセスは、ＣＰＵレジスタにアクセスするのに要する時間よりも２倍から３倍遅い。従来のメモリへのアクセスは、普通、ＣＰＵレジスタのアクセス時間よりも３０倍から５０倍遅い。したがって、プログラムは、変数の値が、ＣＰＵレジスタではなくキャッシュ内か従来のメモリ内の

どちらかに記憶される場合、はるかに実行速度が遅くなる傾向がある。キャッシュメモリ内か従来のメモリ内の変数の記憶は、普通、スピルオーバーと呼ばれる。

【００２０】スピルオーバーの影響を回避するまたは最小限に抑えるために、相当の技術上の努力が向けられてきた。例えば、プログラムの初めに使用される変数は、そのプログラムの終わりには使用されないことが多い。したがって、プログラムの初めにのみ使用される変数の値を記憶するように割り当てられたＣＰＵレジスタは、第１の変数が使用されていない場合、プログラムの残部の実行中、異なる変数への割当てによってより効率的に使用される。プログラムの一部の実行中に使用される変数は、プログラムのその部分が実行されている間、生と呼ばれることが多い。プログラムの一部の実行中に生である変数の集合、すなわちそのプログラムの部分の「生変数集合」を決定することによって、ＣＰＵレジスタのより効率的な割当てが可能になる。プログラムの各部分が実行されると、そのプログラムの部分の生集合のメンバーである変数のみをＣＰＵレジスタ内に記憶する必要がある。

【００２１】一般に、グラフ色付けなど、高度な解析プロセスが、プログラム変数の優先順位付けおよびレジスタの割当てに使用される。これらの技法の例は、論文「Ｒｅｇｉｓｔｅｒ Ａｌｌｏｃａｔｉｏｎ And Ｓｐｉｌｌｉｎｇ Ｖｉａ Ｇｒａｐｈ Ｃｏｌｏｒｉｎｇ」ＳＩＧＰＬＡＮ Ｎｏｔｉｃｅｓ，１７（６）：９６－１０５頁、１９８２年６月、Ｐｒｏｃｅｅｄｉｎｇｓ ｏｆ ｔｈｅ ＡＣＭ ＳＩＧＰＬＡＮ １９８２ Ｓｙｍｐｏｓｉｕｍ ｏｎ Ｃｏｍｐｉｌｅｒ Ｃｏｎｓｔｒｕｃｔｉｏｎに提示されている。生変数の優先順位付けおよびＣＰＵレジスタの割当てに使用されるグラフ色付け技法の他の調査は、参照により本明細書の一部となる、前出のＰｒｅｓｔｏｎ ＢｒｉｇｇｓのＰｈＤ．論文「Ｒｅｇｉｓｔｅｒ Ａｌｌｏｃａｔｉｏｎ Ｖｉａ Ｇｒａｐｈ Ｃｏｌｏｒｉｎｇ」に開示されている。

【００２２】様々なプログラム変数優先順位付けプロセスおよびレジスタの割当て技法が存在し、今日使用されているが、これらの技法はすべて、コンピュータプログラムの様々な部分それぞれについて生変数集合を正確に確認することに依拠する。現在、生変数集合を決定するための種々様々な技法が使用できる。しかし、残念ながら、これらの従来の技法は、計算が集中的で、不正確か、またはその両方である。生変数集合を決定するための一つの技法の例は、定義使用連鎖を解析することである。生変数集合を決定するための他の技法は、プログラムの静的単一割当て（「ＳＳＡ」）変換を解析することである。静的単一割当て変換技法は、通常、定義使用連鎖の解析よりもいくつかの利点を有すると考えられる。一般に、ＳＳＡ変換は、解析中のコンピュータプログラムのサイズが増大するにつれて、定義使用連鎖よりも遅い速度でサイズが大きくなる傾向がある。したがって、ＳＳＡ変換に基づくプロセスを使用する最適化コンパイラは、定義使用連鎖に関連するプロセスに依拠するものよりも、動作が迅速であり、メモリの消費が少なくなる傾向がある。値番号付け、無意味なコードの削除、一定の伝搬など、一般に最適化コンパイラによって実行されるいくつかの他の機能も、定義使用連鎖タイプのプロセスに基づくプロセスの代わりにＳＳＡ変換を使用すれば、より効率的に実行できる。しかし、残念ながら、ＳＳＡ変換は、ＳＳＡ変換に基づく逆方向データフロー解析を実行することによって生変数集合を決定する場合、必ず不正確な結果が得られる。本発明は、ＳＳＡ変換に関連する方法論に基づく生変数集合を決定するより正確かつ効率的な方法を提供する。

【００２３】次に、図４を参照すると、代表的なレジスタ割当て部７０の編成が示されている。これは、図３のブロック５０に対応する。レジスタ割当てルーチン７２へのエントリ上に「制御フローグラフ」技法を使用して「干渉グラフ」を構成する（７４）。「干渉グラフ」構成について以下に詳細に説明する。「フローグラフ」は、そのノードが目的プログラムの「基本ブロック」である有向グラフである。「基本ブロック」は、制御の流れがブロックの初めに入り、停止せずにまたは終わりを除いて分岐する可能性なしにブロックの終わりから出る一連の連続的なステートメントである。引き続き図４で、干渉グラフを構成した（７４）後、干渉グラフを色付けする順序を選択する（７８）。次いで、干渉グラフ中で確認された「仮想レジスタ」を「物理レジスタ」集合内にマップする試みを行う。このマッピングプロセス中、レジスタ内にマップできない変数は、メモリに「スピル」するためにフラグ付けする。仮想レジスタの確認された集合のマッピングの「成功」は、二つの隣接するノードが、可能なすべてのノードの対について異なる色（すなわち実際のレジスタ）を有するグラフとして定義される。したがって、図４では、「成功」についてのテスト８０を行い、成功８２であれば、すべてのレジスタが変数の衝突なしに目的プログラムに割り振られると、レジスタ割当て部ルーチンが励起される（８４）。「成功」でなければ（８８）、前に「スピル」されるようにマーク付けされた変数をメモリに割り当てし（８６）、レジスタ割当てルーチンをスピルされた変数なしに再び開始する（７４）。この反復手順は上記で定義された「成功」が達成されるまで続く。

【００２４】次に、図５を参照すると、干渉グラフを構成する（９０）手順が記載されている。このルーチンへのエントリ９２上で第１の基本ブロックが識別される（９４）。基本ブロックの終わりから始めて、ブロックの終わりの生変数集合内にある仮想レジスタを決定する（９６）。基本ブロックの終わりからの各命令がブロッ

クの開始と逆方向に働く場合、命令と現在の生変数集合とによって定義される各仮想レジスタ間にエッジを生成する（９８）。命令によって「定義」される各仮想レジスタを生変数集合から除去し（１００）、命令によって「使用」される各仮想レジスタを生変数集合に「付加」する（１０２）。このプロセスをブロック内の各命令に対して実行し、ブロック内に命令がなくなったら（１０６）、完了するブロックがまだあるか次のブロックを確認する（１１０）。すべてのブロックをこのように処理したら（１１２）、この目的手順についての干渉グラフが完了する。この段階で、干渉グラフを使用して、図４を参照して上述したように、目的手順内の仮想レジスタをＣＰＵの物理レジスタ内にマップする試みを行う。本発明の好ましい実施形態では、各基本ブロック（図５のブロック９６）の終わりに生変数の集合を見つけるプロセスにおける改善を確認する。この改善は、プログラムのグラフ形式がＳＳＡ形式内にある場合、レジスタ割当て／マッピングの故障を訂正するように設計される。そのような故障の結果、いくつかの変数が、その必要がない場合にメモリにスピルされることになる（したがって目的プログラムの動作時間が遅くなる）。これらの改善は、図６に示す各ブロックの生集合内の変数を見つけるプロセス１２０の文脈で述べられている。

【００２５】次に、図６を参照すると、図５のブロック９６のプロセスの詳細な図が示されている。このプロセスへのエントリ１２２上でファイ関数（Φ関数）の引数の使用を備えたダミーブロックが制御フローグラフに付加される（１２４）。これらのダミーブロックを、Φ関数ブロックの前に、Φ関数に結合されるエッジに付加する。これらのΦ関数について以下に詳細に説明する。次に、各ブロックの初めの生集合を空集合に初期化する（１２６）。この手順についての全体的な制御である大域変化フラグをＦＡＬＳＥに初期化する（１３０）。次いで、目的プログラムの終わりから始めて、目的プログラム内の各基本ブロックに逆方向に働いてプロセスが、ブロックの終わりで生である変数を見つける。このプロセスは、手順１３１内の最後のブロックを得ることによって始まる。目的ブロックの終わりに生である変数の集合の現在の近似値を決定するために、プロセスは、制御フローグラフ中の目的ブロックの継承者である各ブロックの初めに変数の集合を結合する（１３２）。次いで、基本ブロック内の最後の命令１３３からブロック内の最初の命令まで働いて、命令がダミーブロック内で「使用」であれば、「使用された」変数（仮想レジスタ）を生集合に付加する（１３４）。命令が「Φ関数」であれば、Φ関数によって定義される仮想レジスタを生集合から引き（１３６）、Φ関数の引数は生集合には付加されない。他のすべての命令に対して、「定義」された仮想レジスタを引き、仮想レジスタが「使用」されている場合はそれを生集合に付加する（１３８）。処理されたばかりの最新の命令がブロック内の第１の命令である場合（１４４）、これは、ブロックが完了したことを意味する。生集合を検査して、このブロックの処理の初めから変化したかどうかを確認する（１４６）。そうであれば（１４８）、大域変化フラグをＴＲＵＥに設定し（１５０）、制御がブロック（１５４）に行く。生集合が変化しなければ（１５２）、プロセスに対して基本ブロックがまだあるかどうかに関して決定を行う（１５４）。そうであれば（１５６）、プロセスが他のブロック（１６０）を得て、このブロックについてのプロセスをブロック（１３２）から始めて繰り返す。現在のブロックに関するこのプロセスループは、このブロックについての生変数の集合の決定が行われる時点で生集合が変化しなくなるまで（１５２）続く。基本ブロックがまだあれば（１５６）、他のブロックのポインタを得て（１６０）、プロセスをブロック（１３２）から始めて繰り返す。すべてのブロックが完了したら（１５８）、変化フラグをテストし（１６２）、ＴＲＵＥであれば（１６８）、プロセスをブロック（１３０）から再び繰り返す。変化フラグがＦａｌｓｅであれば（１６４）、生集合の決定は完了である（１６６）。次に、このプロセス、および変数がどのように変化し、ダミーブロックとΦ関数ブロックがどのように使用されるかのプロセスについて以下に詳細に説明する。

【００２６】図７を参照すると、コンピュータプログラムの例示部分についての制御フローグラフ２１０が示されている。本発明の一つの態様によれば、生変数の集合は、この部分と、プログラムのあらゆる他の部分とについて決定される。これらの生変数集合を決定してしまえば、ＣＰＵレジスタのより効率的な割当てを実行できる。例えば、図７に示されたプログラム部分の生変数集合の内にない変数の値をプログラムのこの部分の実行中にＣＰＵレジスタ内に記憶する必要はない。生変数集合を決定するために、最初に、図７の制御フローグラフ２１０中に示されたプログラム部分を基本ブロックの集合に分割する。上述のように、基本ブロックは、基本ブロック内のコードの各行が、そのブロック内のコードの第１の行が実行される場合に必ず実行されるという特性を有するコンピュータプログラム内のコードの一部である。基本ブロックは、本質的にプログラムの自立部分であり、恐らく、ブロック内の最後の行を除いて、プログラムの他の部分への分岐命令を含まない。

【００２７】制御フローグラフ２１０によって示されたプログラム部分は、すでにブロック２１２、２１４、２１６、２１８、２１９、２２２および２２４など、基本ブロックに区分されていると考えられる。説明の目的のために、図７に示される基本ブロック２１２～２２４のいくつかは、Ｘなどの変数の様々な例示的な定義および用法を含む。したがって、基本ブロック２１２は、初期定義Ｘ＝Ｙを含み、一方、基本ブロック２１９は、式Ｘ

＝Ｘ＋１の変数の定義と用法の両方を含む。他の基本ブロックは、様々なコンピュータプログラミング言語で一般に使用される形式の例示的な分岐型命令を含むと考えられる。例えば、基本ブロック２１４は、ブロック２１６～２２２を含むループを形成すると考えられる。基本ブロック２１４内の条件「ｗｈｉｌｅ」文と、基本ブロック２１６内の条件「ｉｆ」文はどちらも、それらのそれぞれの条件付き引数に依存する制御フロー分岐を提供する。当業者は、本願に開示したプロセスおよび装置は、いくつか多い変数の定義および使用を含む基本ブロックを有するソフトウェアプログラムにも応用できることを理解するであろう。図７に示される例示的なプログラム部分に示される単一の変数は、単に例示を目的とするものであり、本発明を制限するものと考えるべきではない。

【００２８】本発明の一つの態様によれば、図７に示される基本ブロック２１２～２２４についての生変数集合を決定するプロセスにおける他のステップは、制御フローグラフ２１０に静的単一割当て（「ＳＳＡ」）変換を適用することである。この変換の一部として、新しい変数を参照するために、変数の定義に関する各ステートメントを名前変更する。図８には、図７に示される制御フローグラフ２１０によって表される同じプログラム部分についての制御フローグラフ２２０が示されている。図８の制御フローグラフ２２０では、変数Ｘを含む各定義文は、名前変更されている。例示の目的のために、変数Ｘの用語を保有しているが、変換によって付与される新しい変数名を参照するために、下付きの番号を付加してある。したがって、例えば、基本ブロック２１２内の定義Ｘ＝Ｙは、X_1＝Ｙに名前変更される。同様に、基本ブロック２１９内のステートメントＸ＝Ｘ＋１は、X_4＝X_2＋１に名前変更される。

【００２９】残りの変数とともに、特殊なファイ関数ステートメントもプログラム変換に付加される。これらのファイ関数は、同じ変数の二つまたはそれ以上の異なる定義が制御フローグラフ中で収束する制御フローグラフ２２０中に含まれる。ファイ関数は、変数の新しい「定義」例であり、ファイ関数の引数は、ファイ関数に通じる変数の異なる定義を含む。例えば、基本ブロック２１４は、Ｘの定義中のそのような収束を表す。すなわち、基本ブロック２１４で、ステートメント「ｗｈｉｌｅ（Ｘ＜１０）」内のＸの定義は、基本ブロック２１２によって提供されるＸの定義から得られるか、あるいは基本ブロック２２２によって提供されるＸの定義から得られる。したがって、ファイ関数は、基本ブロック２１２と２１４の間にある基本ブロック２２６として制御フローグラフ２２０中に挿入される。ファイ関数を付加することによって与えられた変数Ｘの新しい定義例を表すために、追加の添数付き変数「X_2」を提供する。

【００３０】ファイ関数の引数は、ファイ関数に導く変数の異なる値から得られる。したがって、基本ブロック２２６でのファイ関数の引数のうちの一つは、基本ブロック２１２でのX_1の定義から得られるX_1である。しかしながら、基本ブロック２２６でのファイ関数の引数の他の部分は、基本ブロック２２２から得られる。図８から分かるように、基本ブロック２２２も、変数Ｘの異なる定義の収束を表す。すなわち、基本ブロック２２２でのＸの定義は、定義「X_3＝１０」による基本ブロック２１８からか、定義「X_4＝X_2＋１」による基本ブロック２１９から得られる。したがって、図８に詳細に示すように、ファイ関数も、基本ブロック２２２の直前の基本ブロック２２８に挿入され、新しい添数付き変数「X_5」を備える。この新しい基本ブロック２２８内の添数付き変数X_5は、基本ブロック２２６内のファイ関数の引数の第２の最後の部分を提供する。ここで、基本ブロック２２６内のファイ関数は、「X_2＝～（X_1、X_5）」のように完全に表すことができる。また、基本ブロック２２８でのファイ関数の引数は、基本ブロック２２８で収束するＸの異なる定義から、すなわち基本ブロック２１８と２１９から得られる。したがって、基本ブロック２１８内の定義X_3＝１０と、基本ブロック２１９内の定義X_4＝X_2＋１とは、基本ブロック２２８でのファイ関数の引数を与え、したがってこれをX_5＝Φ（X_3、X_4）のように完全に表すことができる。

【００３１】制御フローグラフ２２０にファイ関数を付加すれば、図７の制御フローグラフ２１０によって表されるプログラム部分の単一静的割当て変換は完全になる。しかしながら、この変換を使用して生変数集合を決定する場合、やはり不正確な結果が得られ、実際に生でない生集合内の変数の数が過剰であることが分かっている。本発明の他の態様によれば、コンピュータプログラムの単一静的割当て変換は、ファイ関数の引数内の添数付き変数の「使用」を表す制御フローグラフ中に基本ブロックを付加することによってさらに変更される。そのような「使用」ブロックはそれぞれ、ファイ関数の引数内で使用される変数のうちのただ一つの使用を含む。これらの使用ブロックは、ファイ関数と、ファイ関数で制御フローグラフ内に収束する変数の異なる定義との間で制御フローグラフに付加される。

【００３２】図９を参照すると、本発明による図８の制御フローグラフ２２０に付加された使用ブロック２３３、２３４、２３６および２３８を含む制御フローグラフ２３０が示されている。例えば、ブロック２２８のファイ関数は、その引数内に変数X_3とX_4を有する。したがって、使用ブロック２３３と２３４は、ブロック２２８のファイ関数と、それぞれブロック２１８と２１９での定義X_3とX_4との間で制御フローグラフ２３０に付加される。したがって、変数X_4の使用ｆ（X_4）を有するブロック２３３は、その引数内にX_4を有するブロック２２８のファイ関数と、ブロック２１９での変数X_4

15

（すなわち$X_4=X_2+1$）の定義との間で制御フローグラフ２３０に付加される。同様に、変数X_2の使用ｆ（X_3）を有するブロック２３４は、（その引数内にさらにX_3を有する）ブロック２２８のファイ関数と、ブロック２１８での変数X_3（すなわち$X_3=10$）の定義との間で付加される。

【００３３】ブロック２２６で制御フローグラフ２２０内に他のファイ関数があるので、追加の使用ブロック２３６と２３８は、制御フローグラフ２３０に、ブロック２２６のファイ関数とそのファイ関数の引数内の変数の定義との間に付加される。変数X_1の使用ｆ（X_1）を有するブロック２３６は、ブロック２２６でのファイ関数Φ（X_1、X_5）と、ブロック２１２での変数X_1の定義との間に挿入される。同様に、変数X_5の使用ｆ（X_5）を有するブロック２３８は、ブロック２２６のファイ関数Φと、ブロック２２８での変数X_5の定義との間に挿入される。

【００３４】使用ブロック２３３～２３８を付加すれば、制御フローグラフ２３０上で逆方向データフロー解析を実行して、グラフ中の基本ブロックのそれぞれについて生変数集合を決定できる。この解析は、反復プロセスとして実行することが好ましい。各反復では、プロセスは、基本ブロックの終わりで開始される。この解析を実行する場合、基本ブロック内で使用される変数は、そのブロックについての生変数集合に付加される。しかしながら、変数が同じ基本ブロック内でその後定義される場合、変数は、生集合から除去される。本発明の他の態様によれば、ファイ関数の引数内で変数を使用しても、その変数が生集合に付加されない。

【００３５】図１０Ａには、逆方向データフロー解析の第１の反復を行った後の、図９の制御フローグラフ中の基本ブロック２１２～２３８のそれぞれについての生変数の集合が示されている。例えば、基本ブロック２１２について考えてみると、変数Ｙは、Ｙが式X_1＝Ｙに使用されているので、基本ブロック２１２の生集合に付加される。しかしながら、変数X_1は、基本ブロック２１２内で定義されているだけなので、基本ブロック２１２の生集合には付加されない。同様に、基本ブロック２１９の生変数集合は、この変数がブロック２１９内で使用されているのでX_2を含むが、この変数は基本ブロック２１９内で定義されているだけなので、X_4は含まない。基本ブロック２１６、２１８および２２２は、これらのブロックはそれぞれ変数の使用を含まないので空の生変数集合を有すると考えられる。基本ブロック２２６、２２８内の変数の唯一の使用は、ファイ関数の引数内にあり、したがって本発明によれば無視される。したがって、基本ブロック２２６および２２８の生変数集合も空である。基本ブロック２３３～２３８は、上述のように、様々な変数の使用を表すために制御フローグラフ２３０に明確に付加された。これらの基本ブロックそれ

16

ぞれの生集合（図１０Ａに図示）は、関連する変数のそれらの明確な使用を反映する。制御フローグラフ２３０中の残りの二つの基本ブロック、ブロック２１４と２２４は、それぞれ、図１０Ａに詳細に示されているように、変数X_2の使用を含む。

【００３６】本発明によれば、逆方向データフロー解析のプロセスは、反復して実行することが好ましい。本発明の逆方向データフロー解析の第２の反復では、考慮中の基本ブロックの直後の基本ブロックの生変数集合も、生変数集合解析で考慮される。これらの後続のブロックは、通常、子ブロックと呼ばれ、子ブロックの直前の基本ブロックは、通常親ブロックと呼ばれる。したがって、子ブロックの生変数集合の結合も、プロセスの第２の反復で親ブロックの生変数集合の決定に含まれる。さらに、この解析は、最後の基本ブロック、ブロック２２４で開始し、制御フローグラフを逆方式に進行し、図９に示される制御フローグラフ２３０の第１の基本ブロック、ブロック２１２で終了することが好ましい。

【００３７】図１０Ｂには、本発明の逆方向データフロー解析の第２の反復の後の、制御フローグラフ２３０中の基本ブロック２１２～２３８それぞれの生変数集合が示されている。制御フローグラフ２３０の終わりの基本ブロック２２４から始めて、このブロックの生集合は、ブロック２２４に子ブロックがなくなったので、不変である（X_2の使用のみを含んだままである）。基本ブロック２１４の生変数集合も不変である。変数X_2は、「ｗｈｉｌｅ」ステートメント（$X_2<10$）の引数内でこの変数を使用するので、プロセスの第１の反復の後に、すでにブロック２１４の生変数集合内にある。X_2がすでにブロック２１４の生変数集合内にあるので、ブロック２１４に対する子ブロックであるブロック２２４の生集合内でこの変数を考慮することによって変化は生じない。

【００３８】図１０Ａと図１０Ｂを比較すると、さらに、明らかに異なる理由からではあるが、基本ブロック２２６の生変数集合も不変であることが分かる。基本ブロック２１４は、ブロック２２６の子ブロックである。したがって、第２の反復で、基本ブロック２１６の生集合内の変数X_2は、ブロック２２６の生変数を決定する際に考慮される。したがって、変数X_2は基本ブロック２２６の生変数に最初に付加されるが、その場合ブロック２２６もファイ関数の式X_2＝Φ（X_1、X_5）中の変数X_2の定義を含むので、その後除去される。

【００３９】基本ブロック２３８についての考察では、ブロック２３８の子ブロック２２６は、空の生変数集合をまだ有しているように見えた。したがって、親ブロック２３８の生変数集合は不変である。しかしながら、基本ブロック２２２の生変数集合は、プロセスのこの第２の反復で変化する。第１の反復では、基本ブロック２２２は、空の生変数集合を有する。基本ブロック２２２の

子ブロックである基本ブロック２３８の生集合の付加により、変数X_5がブロック２２２の生集合に付加される。

【００４０】基本ブロック２３８の生変数集合についての考察も、本発明の逆方向データフロープロセスを例示する際に役立つ。第１の反復では、ブロック２２８内で使用される唯一の変数、変数X_2およびX_4は、ファイ関数の引数であり、したがって無視される。変数X_5は、ブロック２２８内で「定義」されるだけなので、これもプロセスの第１の反復でブロック２２８の生変数集合に付加されない。ブロック２２８の子ブロックである基本ブロック２２２の生変数集合を付加すると、プロセスの第２の反復で、変数X_5は、最初にブロック２２８の生変数集合に付加されるが、その場合「使用」のために生集合に最初に付加された変数は、基本ブロック内で「定義」されなければ、次いで削除されるので、生集合から除去される。データフロー解析は逆方向に実行されるので、基本ブロック２２８内の変数X_5の定義は、子ブロック２２２の生変数集合を付加することによって行われる変数X_5の付加の後で起こる。引き続いて、基本ブロック２１８の生変数集合は不変であり、プロセスの第２の反復で空集合である。基本ブロック２３４の生変数集合内の変数X_2は、最初にブロック２１８の生集合に付加されるが、ブロック２１８内の変数X_2を後で定義するために次いで除去される。変数X_4も同様に（ブロック２１９の子ブロックであるブロック２３３の生集合内にあるので）、最初に基本ブロック２１９の生集合に付加されるが、ブロック２１９内の変数X_4を後で定義するために次いで除去される。

【００４１】図１０Ａと図１０Ｂに示される様々な基本ブロックの生変数集合を比較すると、生変数集合の変化は、基本ブロック２１６内と基本ブロック２２２内の両方で起こることが分かる。上述のように、変数X_5は、ブロック２２２の子ブロックであるブロック２３８の生集合内の変数がブロック２２２の解析で考慮された場合、ブロック２２２の生集合に付加された。同様に、プロセスの第１の反復では、ブロック２１６内で使用される変数がないので、ブロック２１６の空集合が得られた。しかしながら、第２の反復では、ブロック２１６の子ブロック（ブロック２１８および２１９）の生集合内の変数を考慮した。ブロック２１８は空の生変数集合を有するが、ブロック２１９の生変数集合は変数X_2を含む。

【００４２】本発明の他の態様によれば、制御フローグラフ中の基本ブロックそれぞれの生変数集合を決定するステップは、結果が不変になるまで反復して繰り返される。後の各反復では、子ブロックの生変数集合の結合は、親ブロックの生集合を決定する際に再び考慮される。上記の例では、ブロック２１６およびブロック２２２の生変数集合は、第１の反復と第２の反復の間で変化した。したがって、プロセスは、第３の反復で繰り返される。図１０Ｃには、上記の逆方向データフロープロセスの第３の反復を実行した後の、制御フローグラフ２３０中の基本ブロック２１２～２３８それぞれの生集合が示されている。図１０Ｂと図１０Ｃを比較すると、基本ブロック２１２～２３８のうちのどのブロックについても第２の反復と第３の反復の間で変化する生変数集合はないことが分かる。したがって、生変数集合を決定するプロセスは完了する。

【００４３】プログラム中の基本ブロックの生変数集合を決定すると、様々なレジスタ割当て技法および変数優先順位付け技法を使用して、スピルオーバーを回避できる。上述のように、スピルオーバーは、プログラムの実行中に操作する必要のある変数の数が、ＣＰＵ内のレジスタの数を超えた場合に生じる。

【００４４】レジスタ割当ての代わりの技法として、ＣＰＵレジスタの数よりも少ないメンバーの生変数集合を有するプログラム部分が最初に実行され、次いでＣＰＵレジスタの再割当てが続いて実行され、その後プログラムの次の部分が実行されるようにプログラムの実行を分割できる。したがって、本発明は、スピルオーバーを回避し、かつターゲットコンピュータプログラムの実行の速度を高めるより効率的なプロセスを提供する。グラフ色付けを必要とするプロセスなど、変数優先順位付けプロセスおよびレジスタ割当てプロセスの実行は、図１に示されるコンピュータによって実行される。この例では、本発明のプロセスに従って決定される生変数集合は、メモリ４内に記憶される。

【００４５】当業者は、本発明の範囲から逸脱することなく、本願に開示した本発明の好ましい実施形態の様々な修正および変更が可能であることを理解するであろう。したがって、本発明の範囲は、上記に述べた特定の発明の実施形態に限定すべきではなく、以下に示す請求項およびその同等物によってのみ規定すべきである。

【図面の簡単な説明】

【図１】　本発明が実施されるＣＰＵと従来のメモリを含む、コンピュータの一部を示す図である。

【図２】　コードオプティマイザの位置を示す代表的なコンパイラを示す図である。

【図３】　コードオプティマイザの大規模編成を示す図である。

【図４】　図３のレジスタ割当て部の編成を示す図である。

【図５】　干渉グラフを構成する場合のステップを表す図である。

【図６】　各ブロックの生集合内の変数を見つけるプロセスを示す図である。

【図７】　コンピュータプログラムの一部についての制御フローグラフの一例を示す図である。

【図８】　単一静的割当て変換を適用した後の、図７に

19

示されるコンピュータプログラムの一部についての制御フローグラフである。

【図９】　本発明のプロセスによる変換を実行した後の、図７に示されるプログラム部分についての制御フローグラフである。

【図１０】　図９の制御フローグラフによって表されるプログラム部分の様々な基本ブロックについての生変数の集合を示すチャートである。

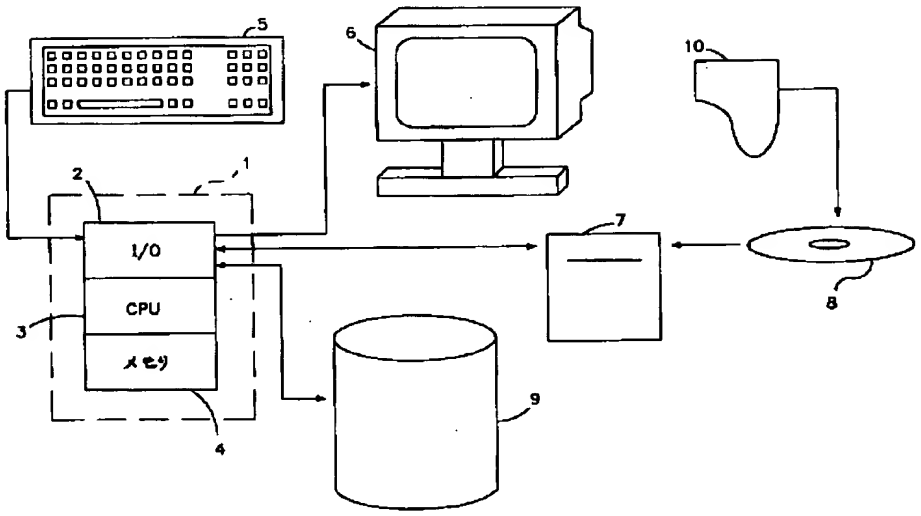
【符号の説明】

１　プロセッサ
２　入出力部
３　中央演算処理装置

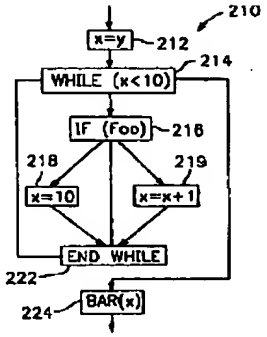
20

４　メモリ部
５　キーボード
６　ディスプレイ装置
７　ＣＤ－ＲＯＭ駆動装置
８　ＣＤ－ＲＯＭ媒体
９　ディスク記憶装置
１０　プログラム
２０　最適化コンパイラ
２４　前端コンパイラ
２６　コードオプティマイザ
２８　後端コード生成器

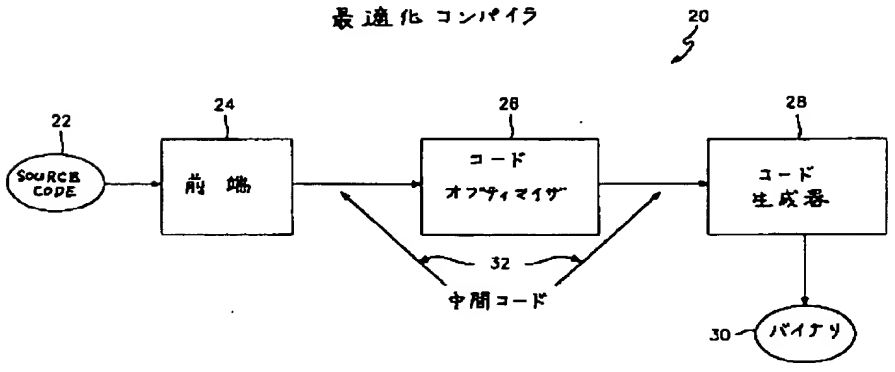
【図１】



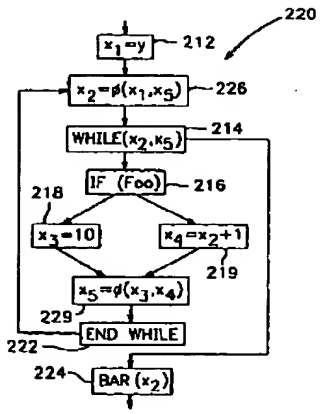
【図７】



【図２】

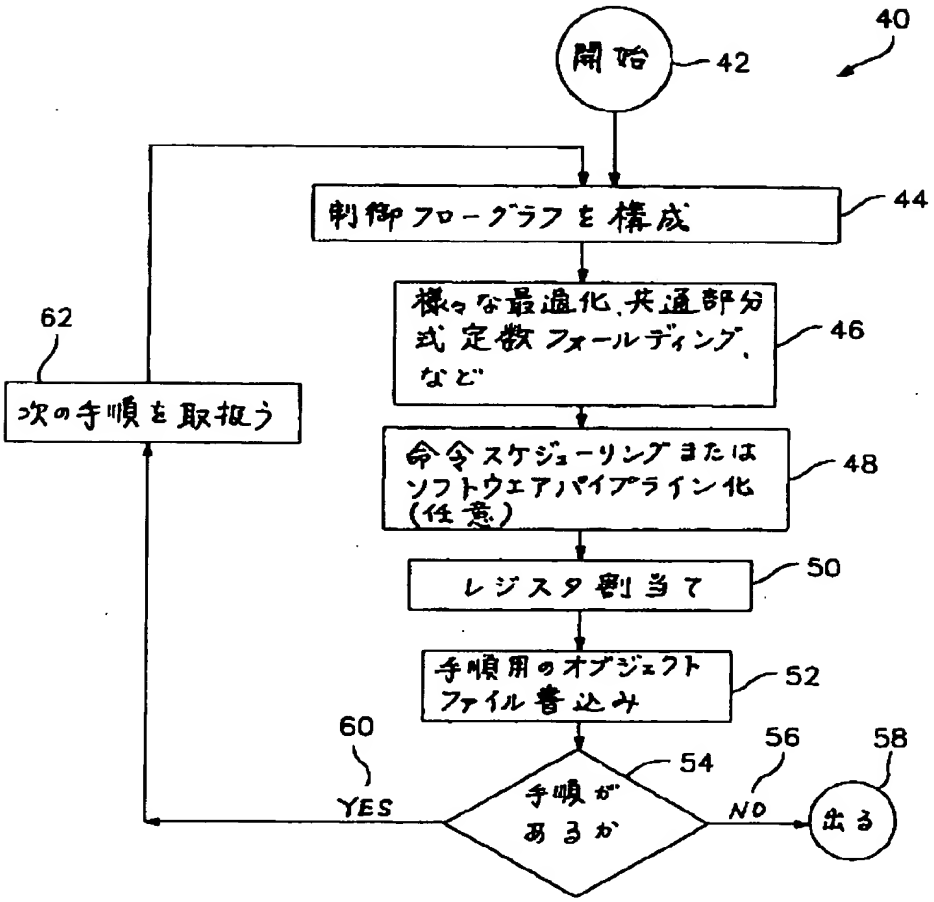


【図８】



【図３】

最適化コンパイラの後端の大規模編成



【図１０】

A

ブロック	生集合
12	y
14	x_2
16	ø
18	ø
19	x_2
22	ø
24	x_2
26	ø
28	ø
33	x_4
34	x_3
36	x_1
38	x_5

B

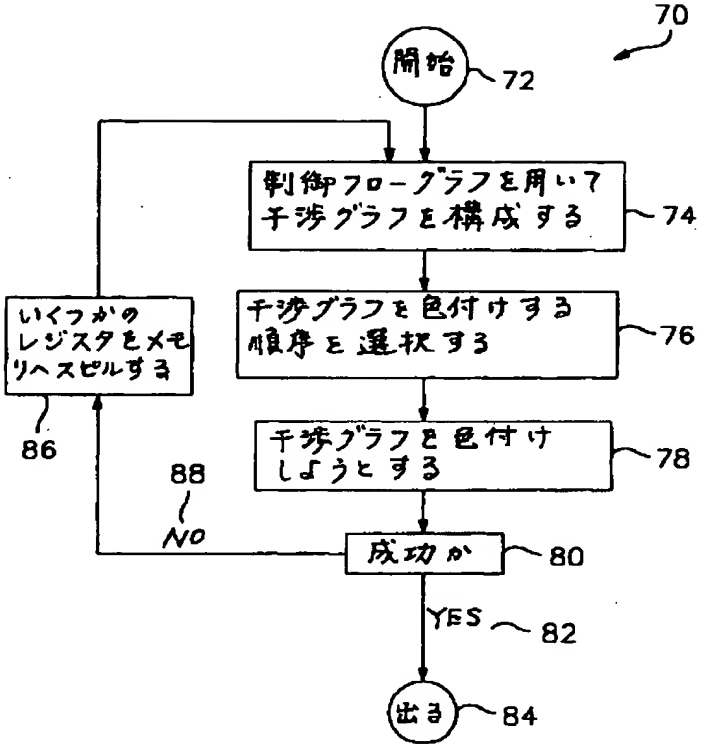
ブロック	生集合
12	y
14	x_2
16	x_2
18	ø
19	x_2
22	x_5
24	x_2
26	ø
28	ø
33	x_4
34	x_3
36	x_1
38	x_5

C

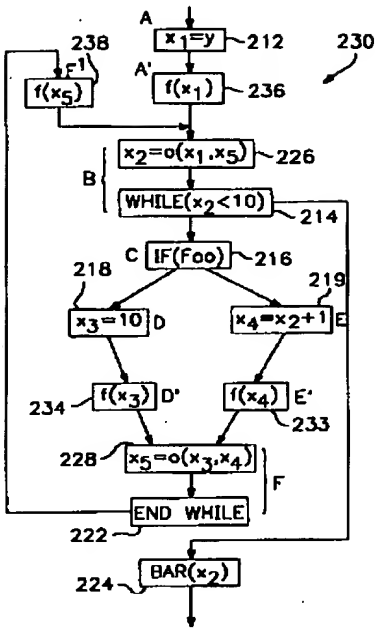
ブロック	生集合
12	y
14	x_2
16	x_2
18	ø
19	x_2
22	x_5
24	x_2
26	ø
28	ø
33	x_4
34	x_3
36	x_1
38	x_5

【図４】

レジスタ割当て編成

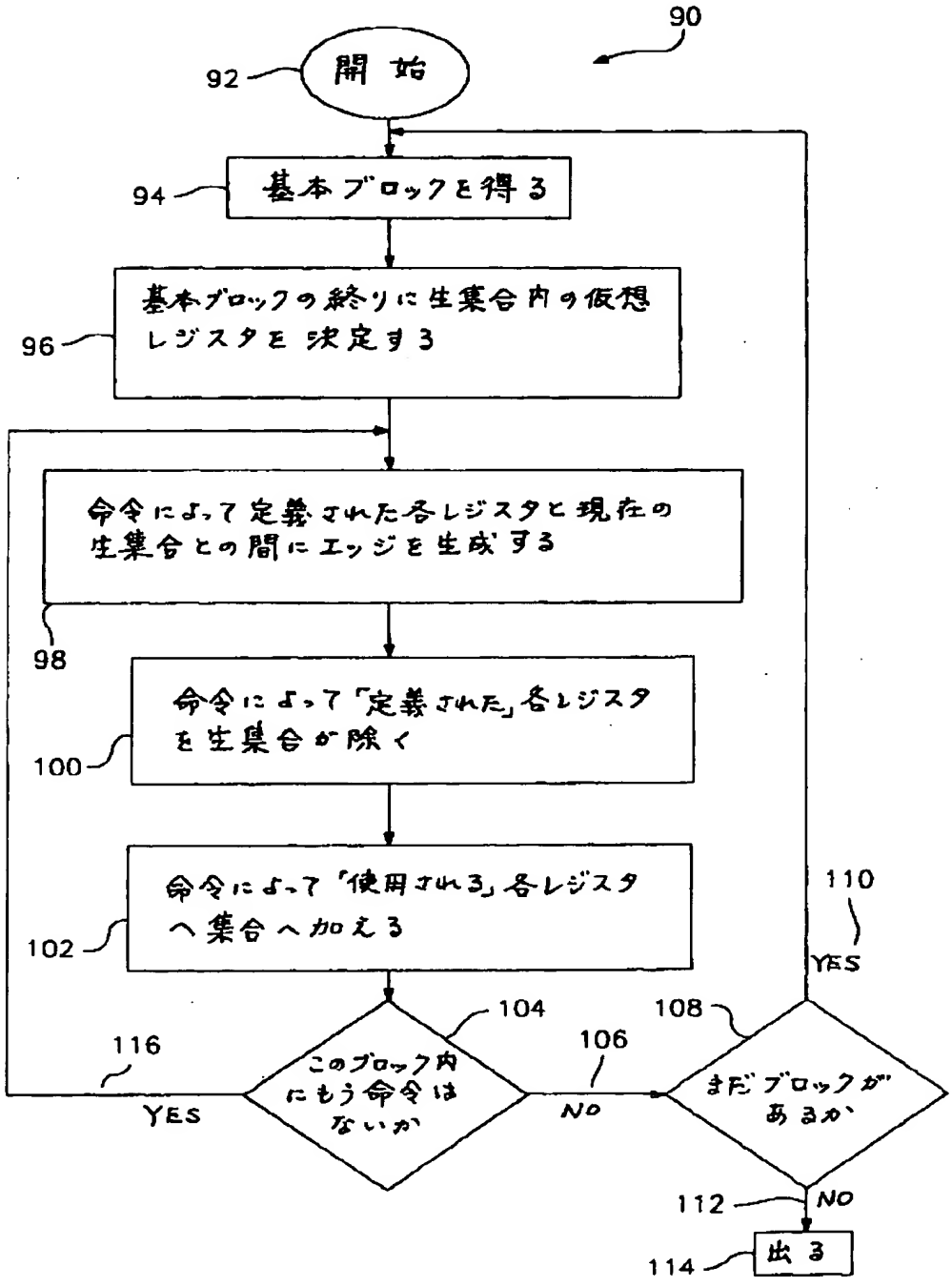


【図９】

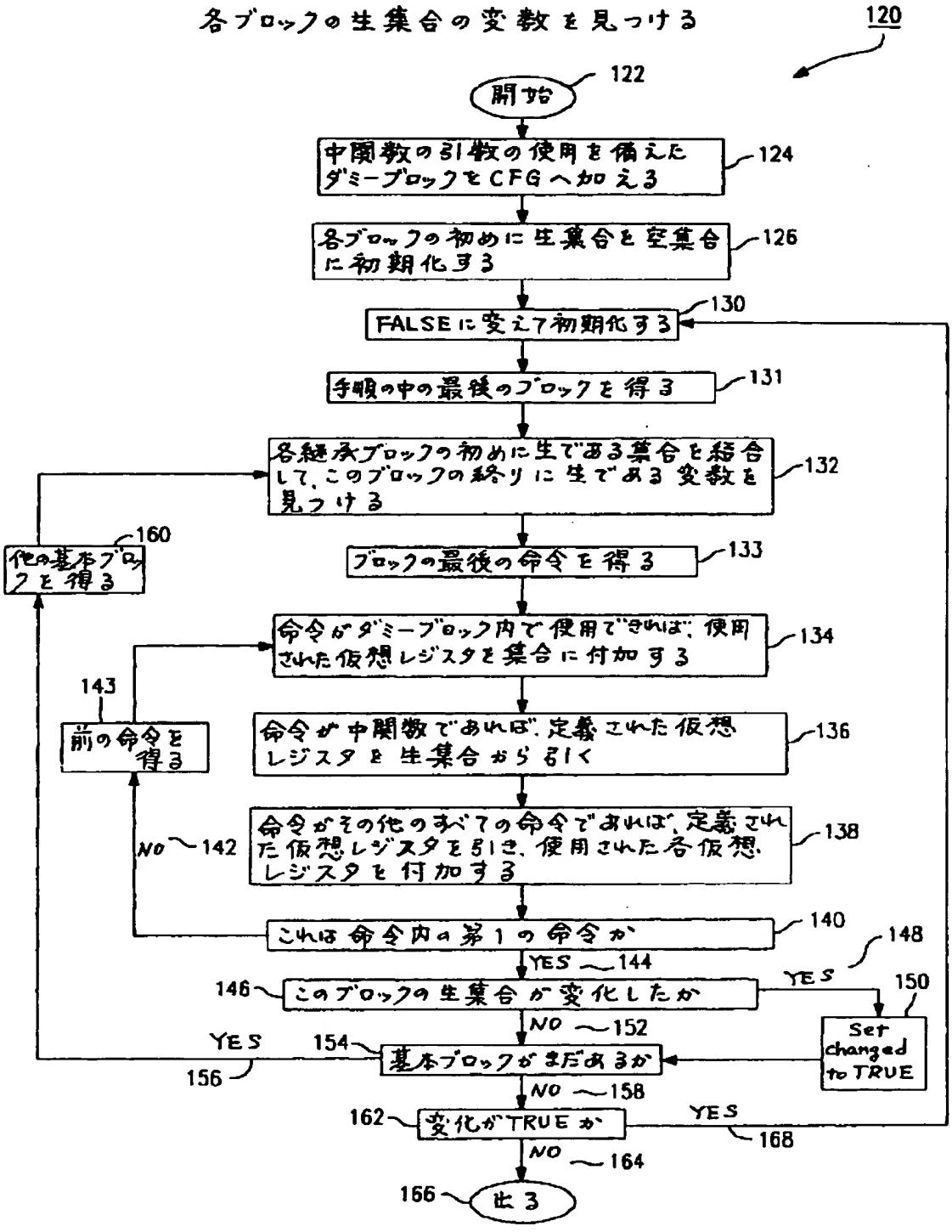


【図5】

干渉グラフの構成


90
92
開始
94
基本ブロックを得る
96
基本ブロックの終りに生集合内の仮想レジスタを決定する
98
命令によって定義された各レジスタと現在の生集合との間にエッジを生成する
100
命令によって「定義された」各レジスタを生集合が除く
102
命令によって「使用される」各レジスタへ集合へ加える
104
このブロック内にもう命令はないか
106
NO
108
まだブロックがあるか
110
YES
112
NO
114
出る
116
YES

【図６】


各ブロックの生集合の変数を見つける
120
開始
122
中関数の引数の使用を備えたダミーブロックをCFGへ加える
124
各ブロックの初めに生集合を空集合に初期化する
126
FALSEに変えて初期化する
130
手順の中の最後のブロックを得る
131
各継承ブロックの初めに生である集合を結合して、このブロックの終りに生である変数を見つける
132
ブロックの最後の命令を得る
133
命令がダミーブロック内で使用できれば、使用された仮想レジスタを集合に付加する
134
命令が中関数であれば、定義された仮想レジスタを生集合から引く
136
命令がその他のすべての命令であれば、定義された仮想レジスタを引き、使用された各仮想レジスタを付加する
138
これは命令内の第１の命令か
140
このブロックの生集合が変化したか
146
基本ブロックがまだあるか
154
変化がTRUEか
162
出る
166
他の基本ブロックを得る
160
前の命令を得る
143
Set changed to TRUE
150
YES
144
NO
142
152
158
164
168
148
156
YES

フロントページの続き

(72)発明者　デビッド・ジイ・シュワルツ
アメリカ合衆国　78212　テキサス州・サン　アントニオ・イー　オルモス　ドライブ・501